

PCT/NZ03/00228

REC'D 17 NOV 2003

WIPO PCT

## CERTIFICATE

This certificate is issued in support of an application for Patent registration in a country outside New Zealand pursuant to the Patents Act 1953 and the Regulations thereunder.

I hereby certify that annexed is a true copy of the Provisional Specification as filed on 14 October 2002 with an application for Letters Patent number 521983 made by MAXIMUM AVAILABILITY LTD.

Dated 5 November 2003.

**PRIORITY DOCUMENT**  
SUBMITTED OR TRANSMITTED IN  
COMPLIANCE WITH  
RULE 17.1(a) OR (b)



Neville Harris  
Commissioner of Patents, Trade Marks and Designs



Patents Form No. 4

Our Ref: JT218083

Patents Act 1953

**PROVISIONAL SPECIFICATION**

**METHOD, SYSTEM AND SOFTWARE FOR JOURNALING SYSTEM  
OBJECTS**

We, **MAXIMUM AVAILABILITY LTD**, a New Zealand company, of 46 Mulgan Way, Browns Bay, Auckland, New Zealand do hereby declare this invention to be described in the following statement:

PT0446353

Intellectual Property  
Office of NZ

14 OCT 2002

**RECEIVED**

## METHOD, SYSTEM AND SOFTWARE FOR JOURNALING SYSTEM OBJECTS

### Field of Invention

The present invention relates to a method, system and software for journaling creation, change and deletion of system objects. More particularly, but not exclusively, the present invention relates to a method, system and software for journaling changes to general OS/400 system objects (including program objects, configuration objects, queues and space/memory mapped objects) in order to replicate these changes on the same or a remote system.

### Background to the Invention

The IBM OS/400 operating system provides journaling of database changes via integrated system functions. This journaling function is primarily orientated toward recording database record level changes for system recovery, commitment control (e.g. to ensure transaction boundaries), auditing, and to support the replay of changes to a replica database (remote or local). Other system objects, such as programs and configuration objects, are not supported by the database journaling function. For these other system objects an independent *System Audit Journal* is maintained. The creation, deletion and changing of system objects may be recorded in the Audit Journal for the primary purpose of providing an audit of activity related to these objects. When viewed with the intention of providing replication of these objects to a remote or local (copy) the Audit Journal has several significant drawbacks, namely:

1. The Audit Journal entries are deposited and made available to other processes on the system only after the associated object activity has been performed. There is no way of trapping the object activity "as-it-happens". This makes additional processing of the object by another system process difficult since the process, which is performing the object activity, is likely to lock and/or use the object before the related Audit Journal entry can be obtained by some monitoring process.
2. The Audit Journal entries are "system-wide" – there is one journal for all objects on the entire system. This requires any monitoring process to

retrieve all of the journal entries even if only a small subset is required for replication.

3. The Audit Journal entries do not contain enough information to perform the associated object activity on another system (or local copy). This requires that any monitoring process must attempt to locate and lock the associated object in an attempt to make a copy prior to another change being performed on the object. If a copy can not be made of the exact state of the object, this state is "lost" and a replication process will be unable to provide the correct object state to a remote (or local copy) system.
4. The only way to serialise Audit Journal entries with a database journal is to attempt to use the journal entry timestamps to merge the Audit Journal entries with the associated database journal entries. This can cause significant overhead in processing the entries for replication. Additionally, when a system has multi-processors the timestamps contained in independent journals may not accurately reflect the exact sequence of operations.

Given these drawbacks, a foolproof method of synchronising system object changes with associated database changes using the Audit Journal has not been available. A method of capturing the content and state of system objects using the same database journal as is used to capture the actual database record level changes would ensure that the database and object changes could be replicated accurately to a remote (or local copy) system.

Since OS/400 system objects are created, changed, and deleted using a standard (finite) set of commands, the most obvious solution to obtaining state information (or making a copy of an object) is to provide replacement commands or to implement a command *exit program*. There are significant drawbacks to both of these approaches

The replacement system commands approach has the following drawbacks:

1. System command parameter interfaces to the associated OS/400 command processing programs can (and often do) change with each release of OS/400. This would cause significant dependency between the replication software and a given release of OS/400.
2. The number and complexity of the commands that would need to be replaced is high (over 150 commands, several with nested lists of parameters). Each command would need it's own replacement processing program as well as significant effort to ensure that each parameter is processed correctly (e.g. as it would be by the original OS/400 command).

The command exit point program approach has the following drawbacks:

1. Neither of the two registered exit points provided by OS/400 allow the associated exit program to be activated after the command has been executed. Therefore, in the case of object creation, change, deletion commands, the exit program is unable to process the resulting object.
2. The QIBM\_QCA\_CHG\_COMMAND exit point is also limited as to the number of exit programs that can be registered. This could prevent some customers from using this exit point if other software uses the exit point.

It is an object of the present invention to provide a method, system and software for journaling system objects which overcomes the above drawbacks or to at least provide the public with a useful choice.

#### Summary of the Invention

According to a first aspect of the invention there is provided a method of journaling changes to system objects comprising the steps of:

- i) substituting a dummy function for a system function;
- ii) executing the system function under operation of the dummy function;
- iii) generating copies of system objects, changed by execution of the system function, for journaling; and
- iv) completing execution of the dummy function;

The dummy function may substitute the system function by having a duplicate calling name, and pre-empting the execution of the system function.

An exit point may be associated with the dummy function and an exit program may be registered for the exit point so that during operation of the dummy function the exit program may be executed. The exit program may handle execution of the system function and capture changes to system objects occurring during such execution. Copies of the changes are generated by the exit program and may be saved to disk or streamed directly to a database system for journaling. The database system may be incorporated with a replication system and may replicate the changes to other local or remote databases.

Messages or exceptions generated by the system function may be captured into a queue.

The dummy function may complete execution by forwarding any messages or exceptions generated by the system function back to the process which called the system function.

The system objects include program objects, configuration objects, queues and space/memory mapped objects.

Changes to system objects include creation, change, and deletion of system objects.

Preferably the system functions are those found on an OS/400 processor.

According to a further aspect of the invention there is provided a method of journaling changes to system objects comprising the steps of:

- i) executing a system function during which changes to system objects occur; and
- ii) journaling changes to system objects during execution of the system function.

One way that changes to system objects may be journaled during execution of the system function is by integrating journaling commands into the code of the system functions.

Another way that changes to system objects may be journaled during execution of the system function is by associating exit points with the system function so that during execution of the system function an exit program may be called to journal the system objects.

According to a further aspect of the invention there is provided a system of journaling changes to system objects comprising:

- i) a processor which executes a dummy function in place of a system function wherein the dummy function executes the system function and generates copies of system objects resulting from system function execution for journaling; and
- ii) memory for use by the processor during execution.

Preferably the processor is an AS/400 processor.

According to a further aspect of the invention there is provided software for effecting the method of the first and second aspects of the invention.

#### Brief Description of the Drawing

The invention will now be described by way of example with reference to the accompanying drawing in which:

Figure 1: shows an illustration of object journaling in relation to a OS/400 Class object.

### Detailed Description of Preferred Embodiments

The following description describes a system object journaling method operating under the OS/400 operating system. It will be appreciated that the method is applicable to other systems with appropriate modifications.

The method uses a combination approach to achieve the desired result; significantly reducing the drawbacks associated with any single approach.

A summary of the steps of the method follows:

1. An exact duplicate of each OS/400 command associated with object changes is made into a new library.
2. The command processing program of each duplicate command is changed to a common supplier provided program which does not need to process any of the command parameters (and therefore is not affected by command parameter changes).
3. A QIBM\_QCA\_RTV\_COMMAND exit point program is registered for each (duplicate) command in the new library. This means that the exit program will be called *before* the associated Command Processing Program (e.g. the program specified in step 2).
4. The new command library is placed in the system library search list, *above* the normal OS/400 system library (QSYS). This causes normal system users and application programs to invoke the commands from the new library rather than their counterparts in the OS/400 system library (QSYS).
5. The exit point program uses the passed command string, to execute the specified command using the original OS/400 command (in the OS/400 system library). The exit point program is then able to perform it's own processing (to capture the object changes) after the OS/400 command has been executed. Any messages sent by the OS/400 command to the exit point program are stored in a temporary queue so that they may be "resent" to the original requestor (e.g. the user and/or application program which submitted the command).



6. The replacement (duplicate) Command Processing Program is then called by OS/400 (when the exit point program has completed). The common (replacement) Command Processing Program simply resends any messages contained in the temporary queue (placed there in step 5).

The method eliminates the need for a custom replacement command processing program for each duplicated command – a single, common program is used for each command. The use of the duplicate version of the commands to attach the exit point program, allows the exit point program to control the processing of the actual system command – performing replication activities both before and after the associated object is created/changed/deleted.

Referring to figure 1, the invention will be described in relation to journaling of a system object on a primary system for replication to a remote journal where the change to the system object is the creation of a OS/400 *Class* object by a user application.

The user application runs with a system library search list that places (step 1) the duplicate command library (MAXSYS) above the operating system command library (QSYS).

This causes the unqualified CRTCLS (Create Class Description) command to bind (step 2) to the MAXSYS library version of the command at run-time.

The replication program MXIICARTET is registered as an exit program (using the system defined QIBM\_QCA\_RTV\_COMMAND exit point) for the MAXSYS version of the CRTCLS command. This causes the MXIICARTET program to be called (step 3) BEFORE the command's *Command Processing Program* is called. The exit point interface passes the full command string text, as specified by the user application, to the exit program.

The exit program performs any required pre-processing to determine if the associated object is defined for replication. The system version of the specified command (in this case, the CRTCLS command) is executed (step 4) by the MXIICARTET program and any messages generated by the system command are stored in a temporary queue.

The affected object (the new class description in this example) is saved (step 5) to a temporary *save file* by the MXIICARTET program.

The temporary *save file* is then copied (step 6) to a temporary OS/400 Integrated File System (IFS) stream file which is journaled to the journal used by the associated replication configuration (e.g. the same journal as used for the database files defined for replication). This effectively stores the save image into the journal.

The generated journal data is then transmitted (step 7) to the remote system(s) for replication using the standard OS/400 remote journal support.

The MXIICARTET program then returns control to its caller (the system exit point), which then calls (step 8) the original Command Processing Program (for the CRTCLS command in the MAXSYS library). For each command in the MAXSYS library, the Command Processing Program is MXCPPNULL.

The MXCPPNULL program retrieves the messages stored in the temporary queue (by the MXIICARTET program), that were generated by the standard system version of the command, and sends (step 9) them to the user application. This allows the user application to process the messages exactly as if it had called the system version of the command directly.

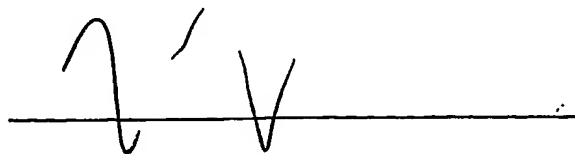
**Definitions**

<b>Class (Class Description)</b>	An object that identifies the run attributes of a job.
<b>Command</b>	A statement used to request a function of the system. A command consists of the command name abbreviation, which identifies the requested function, and its parameters.
<b>Command Processing Program (CPP)</b>	A program that processes a command. This program performs some validity checking and processes the command so that the requested function is performed.
<b>Exit Program</b>	<ol style="list-style-type: none"><li>1. A user-written program that is given control during operation of a system function.</li><li>2. A program to which control is passed from an exit point.</li></ol>
<b>Save File</b>	A file allocated in auxiliary storage that can be used to store saved data on disk (without requiring diskettes or tapes).
<b>System Audit Journal</b>	A journal used by the system to keep a record of security-relevant events that occur.

Where in the foregoing description reference has been made to integers or components having known equivalents then such equivalents are herein incorporated as if individually set forth.

Although this invention has been described by way of example it is to be appreciated that improvements and/or modification may be made thereto without departing from the scope or spirit of the present invention.

MAXIMUM AVAILABILITY LTD  
By its Attorneys  
BALDWIN SHELSTON WATERS

A handwritten signature in dark ink, consisting of a stylized 'M' followed by a 'V' and a horizontal line extending to the right.

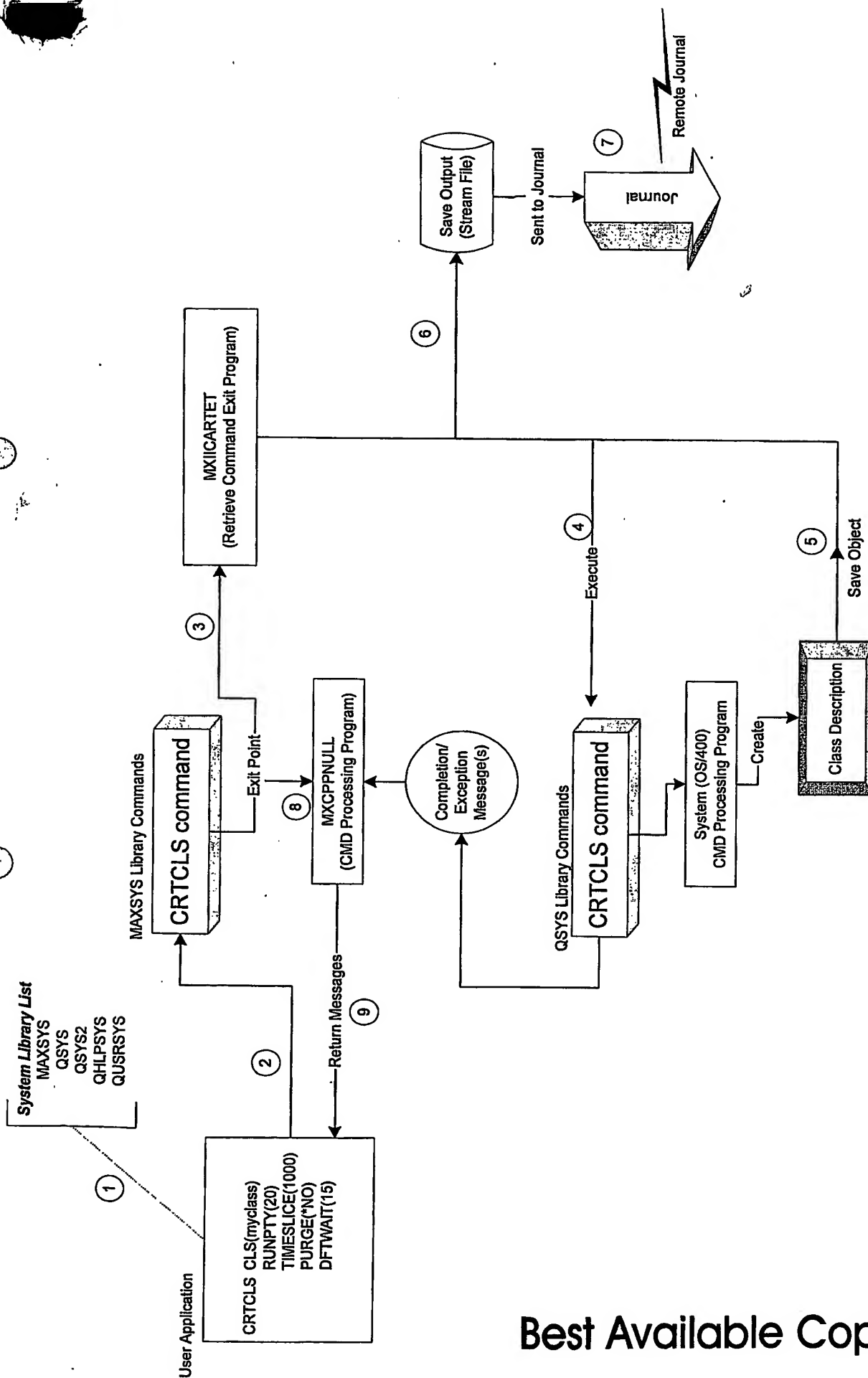


Figure 1